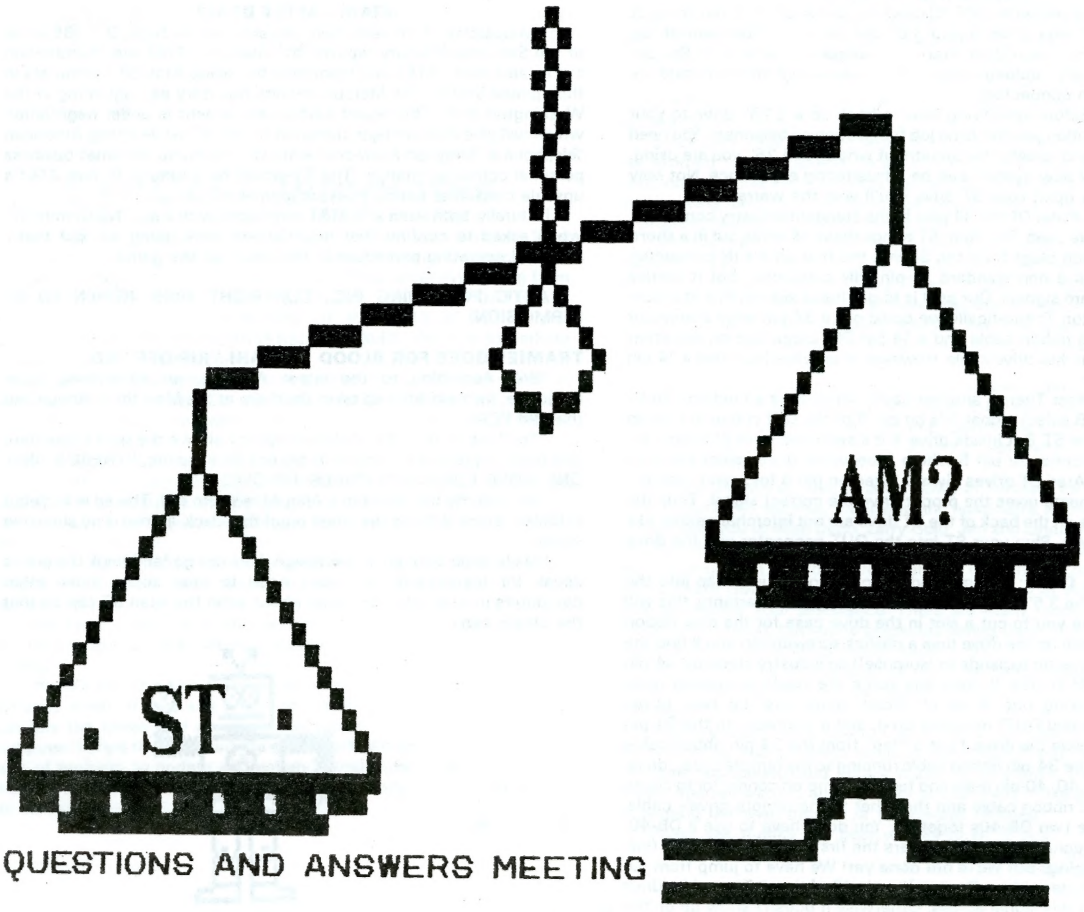
ATARI COMPUTER ENTHUSIASTS

3662 Vine Maple Dr. Eugene OR 97405

OCTOBER, 1985
Mike Dunn, Jim Bumpas & Larry Gold, Editors



QUESTIONS AND ANSWERS MEETING

BUMPAS REVIEWS

Our recent local user group meeting was spectacular enough to deserve comment here. The impressive thing was all the new activity around the Atari 130XE and the entire 8-bit line of Atari computers.

One company demonstrated a voice recognition system which had been developed on the C-64. But the extra memory in the Atari 130XE makes this system work much more efficiently. It's very impressive to see someone speak "red" into a microphone and have the Atari display the color red on the screen.

Kirt Stockwell demonstrated his company's new product, **Mindlink,** a high-powered BBS. Ralph Walden demonstrated his new product, an implementation of the C language with linker and editor. He told us a "benchmark" test he ran was faster on his compiled C for the 8-bit Ataris than with DRI's C language for the STs, and faster than a C language he tested on an IBM PC! The $35 he asks for the disk seems very reasonable.

Another product, **Upgrade**, installs another 64k into your Atari 130XE to give you a total of 192k of memory! The software they provide permits dynamic assignment of the ramdisk to any drive from 1 to 3. They also had an old Atari 800 with memory expanded up to 256k! Imagine. An 8-bit Atari with twice the memory of a MacIntosh and memory equal to the (perhaps soon to appear) C-Amiga!

The ST sort of took a back seat at this meeting. A HEX game was shown, as well as a demo disk of a game called SUNDOG.

STuff

(This will be the section for ST-related material in ACE)

I've been appointed "ST Librarian" for the ACE, and I already have three disks of demo programs, as well as some demo programs (advertisements) of some commercial programs available. If you specify color or monochrome, I'll mail you a disk for $15.

HOW I MADE AN IBM-ST

(An ST with 5.25" Disk Drive)

By David Small, Antic Publishing Inc., Copyright 1985.

Reprinted by Permission.

You can read and write IBM PC disks on your Atari ST if you connect a 40 track 5.25" disk drive to your 3.5" disk drive . . . **You cannot** use the disk drive from your 8-bit Atari . . . **unless** it is an ATR or Percom drive. . . . They are "industry standard" drives which communicate via standard 34-pin connectors.

Warning: Before specifying how to hook up a 5.25" drive to your ST, we must caution you this is no job for electronics beginners. You need to understand and modify the circuitry of whatever 5.25" you are using. Debugging your new system can be a frustrating experience. Not only that, when you open your ST drive you'll void the warranty.

Ribbons & Pins: Of the 34 pins in the standard industry connector, only about 14 are used. The Atari ST brings these 14 wires out in a short, thick cable which plugs from the ST into the first drive's IN connector. This cable uses a non-standard 14-pin DIN connector, but it carries industry standard signals. Our goal is to get these signals to a standard 34-pin connector. Theoretically we could put a 34-pin edge connector on one end of a ribbon cable and a 14-pin DIN connector on the other and we have our disk drive cable. However, in practice I can find a 14-pin DIN connector.

Kinky Wiring: There's another catch. Atari does something kinky with the drive B select signal. It's on pin 6 of the DIN connector when coming from the ST. But inside drive A it's switched from IN connector pin 6 to OUT connector pin 5 where it becomes drive select for drive B. This means Atari ST drives always listen on pin 5 for select, and the daisy chain scheme gives the proper drive the correct signal. Thus the two connectors on the back of the ST drive are not interchangeable, like other Atari drives. Plug your ST into the OUT connector and the drive won't work.

Inside the Drive:: The method I choose to use is to tap into the signals inside the 3.5" ST drive. Besides voiding your warranty, this will probably require you to cut a slot in the drive case for the new ribbon cable. If you open up the drive (use a phillips screwdriver) you'll find the 14-pin DIN connector expands to (surprise!) an industry standard 34-pin ribbon cable. Of course, it does this inside the shield to prevent radio noise from leaking out. A small circuit board has the two 14-pin connectors (IN and OUT) mounted on it, and it connects to the 34-pin ribbon cable inside the drive. I put a "tap" from the 34-pin ribbon cable in the drive to the 34-pin ribbon cable running to my remote 5.25" drive. I then use a DB-40, 40-pin male and female clamp-on connector to clamp one side to the ribbon cable and the other to the remote drive's cable. Then I plug the two DB-40s together. You don't have to use a DB-40. Any clamp-on connector which covers the first 34 pins will work fine.

Pin Swapping: But we're not done yet! We have to jump from pin 6 of the dIN connect (drive B select) to pin 12 of the ribbon cable (drive B select) to get this signal across. Otherwise it doesn't show up on the 34-pin cable. This is easy to do on the bottom of the 3.5" drive's DIN connect board.

Drive B Configuration: Almost done. Now we need to set the remote drive as drive B. Sometimes it's called drive 1 or drive 2, depending on whether the manufacturer numbers drives at 0 or 1. When a drive is idle, a five-volt signal (HIGH) exists on the BUSY line. When the computer wants to access the drive, it pulls down this signal to zero (LOW). When the computer is finished with the drive, it releases the signal and the drive "pulls up" the signal to its original five volts. If two drives are hooked up, only one may contain pull up circuitry because the computer can only pull down five volts. Pull-up circuitry usually is contained in a chip in the drive. And now you are at a point where you must know enough about your 5.25" drive to figure out where the chip is. Since the ST drive A contains all the pull-up termination circuitry we need we must remove termination packs from the remote drive. In the case of my Tandon TM-100-2 drive I also need to deal with the select line termination, since it doesn't go through the resistor pack. I have to clip resistor R14 from my Tandon to get rid of the added termination. Special Note: The ST monitor throws out a lot of magnetism. If you don't keep your drive at least one foot from the monitor, the disk's heads will pick up the monitor's signals and confuse the read data. You'll immediately notice data error if you get your drive close to the monitor. This is good reason to use a fairly long ribbon cable (3 feet or so) [we haven't noticed this problem in-house — ANTIC ED]. *ALL DONE!*

IBM ST: . . . **With** an IBM PC disk in that 40 track drive, . . . click on the B icon. It'll pull up the disk's directory into folders and "text only" files. You'll notice on the top of the window a PC-DOS type of "pathname" consisting of multiple (if needed) folders and a filename. GEM simply turns the concept of pathnames into folder icons and moves you through the path by your actions of selecting, opening, or closing a folder. Of course, you can't run IBM programs because they are written in **languages** which the ST cannot understand. However, you can freely copy and use text files and the data within them. Furthermore, if you write back out from the ST to the PC disk, you'll find an IBM has no trouble reading what you wrote.

ATARI - ATT&T DEAL?

The headline of the lead business story in the Sept. 9, 1985 issue of the San Jose Mercury reports that Atari and AT&T are "hammering out a sales deal." AT&T will reportedly be selling Atari ST computers in their phone stores. The Mercury credits the story as originating in the Washington Post. This report said an agreement is under negotiation which will give Atari a major customer for the ST while giving American Telephone & Telegraph a low-cost entry into the home and small business personal computer market. The ST would be a natural fit into AT&T's upscale consumer phone marketing pipeline.

Naturally, both Atari and AT&T responded with a big "No Comment" when asked to confirm that negotiations were going on. But that's standard operating procedure at this stage of the game.

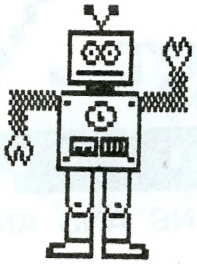
TRAMIEL GOES FOR BLOOD IN ATARI "RIP-OFF" AD

"9/4—According to the latest Adweek, an advertising trade magazine, the next Atari ad takes dead aim at the Mac, the C-Amiga and the IBM PCAT.

"The first ad from Atari's latest agency shows the rival computers and their suggested list prices. A big headline shouts, "THERE'S ONLY ONE WORD FOR THESE PRICES: RIP-OFF."

"Not exactly the old Atari's Alan Alda warm sell. The ad is to debut in Rolling Stone 9/26 as the latest proof that Jack Tramiel is no shrinking violet."

I really hope such an ad campaign does not go far. I think the prices speak for themselves. We don't need to hear about those other computers in Atari ads. Let's hear about what the Atari ST can do that the others can't.



S/T Applications, 10760 Hwy 116, Forestville, CA 95436 is a monthly magazine devoted to the Atari ST. Its first issue is 20 pages and subscriptions are $25 for 6 issues or $50 for 6 issues on disk. The first issue contains a couple of reviews and lists some software coming out for the ST. It also contains some information about the OS.

The editor admits to no experience in producing a magazine, and this first issue is rather poorly photocopied. There is no advertizing, so the pages are full of content — which is sure to become more meaty as time goes on. I hope his reproduction improves quickly

One of the more interesting items in this first issue appears on page 4 as a "Reader Comment":

"For those of you who didn't see Computer Current's Aug 27-Sep 9 issue: On page 43 they tell how some insider at Atari sneaked these four icons in among the control characters as a joke. They make a picture of J.R. "Bob" Dobbs of the Church of the Sub-Genius. He is the invention of art students who wanted to create a bogus religion based on flying saucers, lunatic conspiracy theories and the occult. One of his sayings is "F - 'em if they can't take a joke."

"If you check pages 63 and 65 of your sourcebook for Atari (ST) LOGO, you'll see these icons as chars 28 through 31, if you look carefully. They're just to the right and above the two which make the Atari symbol.

"Study the **following** listing to see the difference between print and type.

```
TO BOB
REPEAT 35 [TYPE CHAR 28 TYPE CHAR 29. PR "REPEAT 35 [TYPE
CHAR 30 TYPE CHAR 31] PR "PR [F - 'EM IF THEY CAN'T TAKE A JOKE.]
END
```

You can also print out the Atari logo with this short routine:

```
TO ATARI
REPEAT 25 [TYPE CHAR 14 TYPE CHAR 15 TYPE CHAR 32]
END
```

Lois Hansen
Oakland, CA

SynCalc Templates

(Synapse, 17 Paul Dr., San Rafael, Ca. 94903.)

The SynCalc Templates are designed to be used with SynCalc. They are 22 of the most commonly used spreadsheet formats. One might have to spend hours just setting up one of these. They cover everything from LINEAR REGRESSION CALCULATIONS, BOND PORTFOLIO EVALUATIONS, to COST ANALYSIS OF PAINTING A ROOM, and KITCHEN MEASUREMENT CONVERSION TABLES. There is something for everyone who uses a computer.

Not only did Synapse cover the gamut in the different types of templates which can be used, but they made them simple to use. The instructions fit into the notebook coming with SynCalc. They just make up another section to refer to very easily. Apart from the fact that the instructions are easy to use and read they cover everything.

First off they start by telling you how to load the program once you have loaded SynCalc. Then they give you an instruction sheet for each template, even the index not only gives the name of the template but tells what it does.

If you use SynCalc then you need this program as even if the templates don't cover what you want they are easy to convert to what you do need.

There is one function of this program which I have not used as yet and that is the increased memory which can be used if you have such a thing.

This is a well thought out program which is a useful addition to the Synapse series. If they add a word processor to this series one could just buy the series and do about everything one might want with a computer. I hope they keep up the good work and bring out more of this type of program to update and enhance their other programs. This is the type of software which will make the computer a more useful tool for all to use.

— LARRY GOLD

ERACE

WALLY'S WORD WORKS

($65, Sunburst Communications, Elementary disk (grade levels: 4-6)

This package adds the elements of whimsy delight and challenge to the practice of basic language skills. Identifying parts of speech is central to the study of sentences.

Wally is a wallaby (a small kangaroo). The player plays by directing Wally over the sentence and picks up words with the arrow key/joystick. Then he drops that word into the correct pocket (i.e.; noun, verb, preposition, etc.). The player also get bonus points for evading the speedy Rovers who are little creatures who chase Wally when he has not picked up a word.

The main menu has 9 categories- Play, demo, Definitions, Instructions, etc.

The package can be tailored to fit the student's needs. It is a neat way to teach the parts of the sentence in a fun way. Available are two more disks in this series: Pocket Pitfalls and Rovers' Revenge for more advanced studies.

As usual, Sunburst sends a bright orange notebook to hold the manual and disks. The instructions are simple to use with ideas.

— Nora Young

FAREWELL

The Ness' have been "unsung heroes" of ACE for a long time. They have spent many hours keeping our program library operating for several years. We all regret their moving on, and wish them the best of luck. The new Program Library chairpeople are **Chuck and Jody Ross, 2222 Ironwood, Eugene, OR 97401** . Thanks again, Ron and Aaron!

— M. Dunn, Co-Editor

SO LONG, FAREWELL, & THANKS

After a year of trying to start a new career, I have decided to return to the airline industry and will be moving (my body only) to Memphis, Tennessee. My heart and home will always be here in Eugene and I will be commuting between the two as often as possible (but not often enough to take care of the library). Aaron will be starting college at Oregon State this fall majoring in the sciences, and will also be unable to care for the library.

Since they have had two years of rest, the duties of the library will be returned to the capable hands of Chuck and Jody Ross. We will certainly miss the early morning phone calls from overseas, the pile of mail arriving daily, and the most able assistance of our postal service in properly folding the disks we receive.

The mail we have received from nearly every corner of the world has made this a most enjoyable task. We are proud to have been involved in the rapid growth of the library, from 17 disks in 1983 to nearly 70 now. However, this could not have been possible without the vast support we have received from our many readers. We have made many friends both for the library and ourselves and we hope to continue to hear from you.

Thanks to the many contributors and correspondents such as Stan Ockers in Illinois, Paul Freeman in Maryland, Ed Slawson in Maine, Wayne Real in Australia, and Les Ellingham in England. The trouble is that if we mention a few, we will omit many others so we will just say THANKS TO ALL OF YOU.

Locally, thanks are due to John Kelley in Portland for his many fine contributions, his warmth, friendship and hospitality. In Eugene, we have to say thank you to Mike Dunn, Larry Gold, Jim and Linda Bumpas, Chuck and Jody Ross, Kirt Stockwell, Dick Barkley, E.J. Knoll, Bob Browning, Bruce Ebling, Ruth Ellsworth, Don Marr and his staff at Computer Palace and to Stacy Goff (wherever you are)! We want all of you to know we truly appreciate your assistance. There are many more names which should be included, but space precludes listing them here. Your support and friendship will always remain dear to us.

We want to say thanks to all of you who started your letters with, "Dear Mr. & Mrs. Ness...", as father and son, we found this quite humorous.

A very special thank you to Carole Ness, wife and mother, for her understanding and patience beyond reasonable limits, while we had computer equipment and library materials scattered throughout the house. Also we want to apologize for occasionally making disks instead of keeping the lawn mowed or the garage cleaned.

Again, our thanks and best wishes to all of you.

— Ron and Aaron Ness

SORTING ROUTINES

IN BASIC AND ACTION

RICHARD GROSZKIEWICZ
SOFTWARE POLISH

Sorting data is an excellent example of something which should be done by a computer and not done by hand. Sorting routines vary widely in complexity and speed of execution. This article discusses several sorting algorithms in both BASIC and ACTION. The actual execution times and source code are given for each algorithm.

OVERVIEW

There are six program listings at the end of this article. Here is a short description of each:

(1) NUM1000.BAS - Creates a test file of 1000 random numbers

(2) NUM1000.ACT - Same as (1), written in ACTION

(3) SORTNUM.BAS - BASIC Bubble and Heap sorts program for numbers

(4) SORTNUM.ACT - Same as (3), written in ACTION

(5) SORTSTR.BAS - BASIC Bubble and Heap sorts program for strings

(6) QUICKSRT.ACT- Uses procedure from ACTION Toolkit to sort numbers

If you compare the code, you can see that precisely the same algorithms were used for both Listings 3 and 4. A LISTed version of the BASIC program in Listing 3 was used to create the ACTION program in Listing 4. This was made easier by the fact that Listing 3 consisted of several sets of subroutines. The hardest part of the conversion was unraveling the spaghetti code between lines 680 and 730 of Listing 3. Some print statements were eliminated in the ACTION versions, since they run very fast.

RESULTS

The following table shows the number of seconds required to sort sets of numeric items. The input file contained random numbers between 1 and 1000, which were produced by the program in Listing 1. The following results are for the sort programs in Listings 3, 4, and 6.

Items 100 999
BASIC
Bubble Sort 115 13,513
Heap Sort 31 471
Machine Language
Bubble Sort 1.60 173
ACTION!
Bubble Sort 1.25 128
Heap Sort .50 7.0
Quick Sort .50 6.0

The following table is similar to the prior table, except that The input file contained **sorted** numbers between 1 and 1000:

Items 100 999
BASIC
Bubble Sort 1.5 5
Heap Sort 32 493
Machine Language
Bubble Sort 11/60 20/60
ACTION!
Bubble Sort 1/60 11/60
Heap Sort .50 7.3
Quick Sort .50 37

As shown above, the Bubble sort is very fast if the data is sorted, while the Quick sort is much slower! The Heap sort is not sensitive to whether the data is sorted or not.

No table was prepared for the sort routines in Listing 4. These routines are similar to those in Listing 3, but they have been modified to sort string variables. The sorting times bear similar relationships to those shown above for the Bubble and Heap sorts. The file DUP.SYS was used as a source of string data that would be constant.

DISCUSSION OF SORTING ALGORITHMS

The programs in Listings 1 and 2 create a test file that can be used by the other programs. This same data was used by each sort routine. Otherwise, some extraneous differences in execution time could result due to non-random numbers!

Bubble Sort - BASIC: this is the simplest of all sort routines. Unfortunately, it is also the slowest, and the most inefficient. An increase in the number of elements by a factor of 10 increases the sort time by a factor of 117! The BASIC version of the bubble sort is shown in lines 500 to 600 of Listings 3 and 5.

Bubble Sort - Machine language: This is approximately 75 times faster than the BASIC version. This routine is from the March 1982 issue of **Compute!** magazine. It is the only published machine language sort for the ATARI I know of which actually works. This routine is contained in the DATA statements from line 6160 to 6330, and the setup is shown in lines 400 to 500 of Listings 3 and 5. The machine language version is quite fast, but it suffers from the same inefficiency as the BASIC version.

The method for calling the machine language bubble sort is clear when sorting strings, as shown in Listing 5. It is trickier when used to sort real numbers, as shown in Listing 4. This is because each real number is stored in six bytes of memory in ATARI BASIC. This is why the record length is declared as six in line 440.

The machine language sort routine must be given the address of the array to be sorted. It is not possible to directly obtain the address of a real variable in ATARI BASIC. Instead, the string variable NUM$(4) is defined in line 1100 directly preceding the array S(999). The address of S(999) is equal to ADR(NUM$)+4, since four bytes are allocated to the string NUM$.

C ie last problem is that all arrays are allocated storage starting at the zeroth element. Since the first array element to be sorted is stored in S(1) instead of S(0), we must define the starting address to pass to the machine language sort as ADR(NUM$)+4+6.

Bubble Sort - ACTION: This version of the bubble sort is 100 times faster then the BASIC version. This is faster then the machine language version of the bubble sort! The reason is ACTION stores the sort items as two byte integers instead of six bytes in memory. The machine language sort actually has to do three times as much work as a result.

Heap Sort - BASIC: This algorithm is slightly more complicated than the bubble sort. It came from the April 1984 issue of **PC TECH JOURNAL** (which contains numerous other routines). The idea of this routine is to pass through the data twice. It builds a "heap" the first time through. This is fairly fast for BASIC, and it is more efficient than the bubble sort. The heap sort starts out four times faster than the bubble sort, and ends up 30 times faster.

Heap Sort - ACTION: This version of the heap sort is roughly 70 times faster than the BASIC version. This is a typical result of the type of speed improvement you can get by programming in ACTION!.

Quick Sort - ACTION: The actual algorithm for the quick sort is not shown in Listing 6. I cheated and used the version contained in the ACTION! Toolkit, and INCLUDEd it in my source code. There is a listing of the quick sort in Microsoft BASIC in the April 1984 issue of **PC TECH JOURNAL[.**

The quick sort is generally accepted as the most efficient sorting algorithm. It is also one of the most difficult to understand. I've included times for this algorithm to show that (a) it is very efficient on unsorted data, but (b) it becomes inefficient if the data is partially sorted already.

SUMMARY/CONCLUSIONS

Each of the sorting methods has strengths and weaknesses:

Bubble Sort (BASIC): Very easy to understand, but too slow.

Bubble Sort (M.L.): Fast for all but the largest arrays, very compact. Does not sort arrays correctly if they contain both negative and positive numbers.

VP RAMBLINGS

It is now October and the year is almost over. In the land of Atari many new things are happening. The one which has me excited is the laser-disk system allowing one to have a disk with say a complete encyclopedia on it and one can access any subject as quick as you can find it in the index, and with pictures. This can revolutionize libraries, and anywhere else where they store information.

The new BBS is running very smoothly now as most of the bugs have been worked out. The only thing we have to do at this point is to change over to 1200 baud. We are working on that and soon we will have it running at that baud. Call the board and see what you think. I want to thank you all for your patience when we were having so much trouble with the board and many of you couldn't get on or your password was lost. We hope those problems and all the others are now gone forever. Remember this system requires you to put in your own PASSWORD and not one which we give you. Anyway, enjoy the board. We hope you will upload your programs to us so we may put them on the board for others to use and enjoy.

As you may have surmised "C" is the coming language for eight and sixteen bit machines, and we are publishing programs in this language. If you want to see more let us know and we can start a separate section devoted just to "C".

This month I have used several of XLEnt's programs to do the newsletter. I hope you like what you see and next month I will have an indepth review of their products.

— Larry Gold

SORT BASIC

```
0 REM ..... FILE: SORTSTR.BAS
10 REM
100 GOSUB 1000:REM WHICH, #, GET DATA
200 GOSUB 2000:REM SORT THE DATA
300 GOSUB 3000:REM PRINT ELAPSED TIME
350 END
400 REM
            M.L. BUBBLESORT
410 REM POKE START/END OF SORT KEY
420 POKE 203,0:POKE 204,4
430 REM REC LENGTH=6, WANT ASCENDING
440 POKE 205,5:POKE 206,0
450 REM NOW MAKE THE USR CALL!
460 A=USR(ADR(SORT$),ADR(S$),N)
470 RETURN
480 REM
            BUBBLESORT
500 K=N
510 T=0
520 FOR I=2 TO K
530 LAST=(I-2)*L+1
540 CURRENT=LAST+L
550 IF S$(LAST,CURRENT-1)<=S$(CURRENT,
CURRENT+L-1) THEN 570
560 T=I:HOLD$=S$(LAST,CURRENT-1):S$(LA
ST,CURRENT-1)=S$(CURRENT,CURRENT+L-1):
S$(CURRENT,CURRENT+L-1)=HOLD$
570 NEXT I:? ".";
575 K=T-1:IF K>1 THEN 510
580 ? :RETURN
585 REM
              HEAPSORT
600 R=N
610 FOR LL=INT(N/2) TO 1 STEP -1
620 HOLD$=S$(L*(LL-1)+1,L*LL)
630 GOSUB 680:NEXT LL:LL=1
640 FOR R=N-1 TO 1 STEP -1
650 HOLD$=S$(L*R+1,L*(R+1))
660 S$(L*R+1,L*(R+1))=S$(1,L)
670 GOSUB 680:NEXT R:? :RETURN
680 J=LL
690 I=J:J=2*J
700 ON 2+SGN(J-R) GOTO 710,720,730
710 IF S$(L*(J-1)+1,L*J)<S$(L*J+1,(L*(
J+1))) THEN J=J+1
720 IF HOLD$<S$(L*(J-1)+1,L*J) THEN S$
(L*(I-1)+1,L*I)=S$(L*(J-1)+1,L*J):GO T
O 690
730 HOLD2$=HOLD$:HOLD$=S$(L*(I-1)+1,L*
I):S$(L*(I-1)+1,L*I)=HOLD2$:? ".";:RET
URN
780 REM
            GETKEY ROUTINE
800 POKE 764,255
810 IF PEEK(764)=255 THEN 810
820 KEY=PEEK(764)
830 POKE 764,255
840 RETURN
880 REM
            TIMER ROUTINE
900 OLDTIME=TIME
910 FLAG=1-FLAG
920 TIME=PEEK(20)+256*(PEEK(19)+256*PE
EK(18))
930 TIME=INT(1000*TIME/60)/1000
940 ELAPSE=TIME-OLDTIME
950 ? "TIME = ";TIME
960 IF FLAG=0 THEN ? "ELAPSE = ";ELAPS
E
970 RETURN
980 REM
          INITIALIZATION ROUTINE
1000 ? CHR$(125)
1010 POKE 712,0
1020 POKE 710,4+16*INT(16*RND(1))
1030 POKE 709,12
1100 DIM TYP$(1),NUM$(4),SORT$(126)
1105 GOSUB 6000
1110 ? :? "Which sort to execute?"
1120 ? :? "(B) Bubblesort"
1125 ? :? "(M) M.L. Bubble"
1130 ? :? "(H) Heapsort":? :?
1140 GOSUB 800
1150 IF KEY=21 THEN TYP$="B"
1155 IF KEY=37 THEN TYP$="M"
1160 IF KEY=57 THEN TYP$="H"
1170 IF TYP$<>"B" AND TYP$<>"M" AND TY
P$<>"H" THEN GO TO 1140
1200 ? :? :? "How many items do you wa
nt to sort"
1205 ? "Max = 999, Minimum = 10"
1210 INPUT NUM$
1220 TRAP 1210
1230 N=VAL(NUM$)
1240 IF N<10 THEN 1205
1250 IF N>999 THEN 1205
1260 N=INT(N):TRAP 34567
1790 L=5
1800 DIM S$(N*L),HOLD$(L),HOLD2$(L)
1805 S$=CHR$(0):S$(N*L)=S$:S$(2)=S$
1810 NUMBER=N*L
1820 ADDRESS=ADR(S$)
1830 IO=1
1840 OPEN #IO,4,0,"D:DUP.SYS"
1850 GOSUB 7000
1900 GOSUB 900
1980 RETURN
1990 REM
              DO SORTS
2000 IF TYP$="M" THEN GOSUB 400
2010 IF TYP$="B" THEN GOSUB 500
2020 IF TYP$="H" THEN GOSUB 600
2900 RETURN
2990 REM
            PRINT RESULTS
3000 GOSUB 900:POKE 766,1
3050 GOSUB 800
3100 FOR I=1 TO N:? I,S$(L*(I-1)+1,I*L
):NEXT I
3900 POKE 766,0:RETURN
6000 REM USR Sort routine relocatable
6010 REM Example assumes records in
6020 REM S$, number of records is N.
6030 REM Need to POKE starting and
6040 REM ending positions (relative)
6050 REM of SORT key plus total recrd
6060 REM length, and ascend vs descnd
6070 REM
6080 REM START  SORT KEY:POKE 203,STR
6090 REM END OF SORT KEY:POKE 204,END
6100 REM REC LENGTH=17  :POKE 205,RL
6110 REM ASC=0, DESC=1  :POKE 206,?
6120 REM
6125 ? "       One moment please..."
6130 FOR I=1 TO 126:READ A
6140 SORT$(I)=CHR$(A):NEXT I
6150 RETURN
6160 DATA 104,104,133,217,104,133,216
6170 DATA 104,133,209,104,133,208,169
6180 DATA 0,133,218,133,207,162,1
6190 DATA 165,216,133,214,165,217,133
6200 DATA 215,24,165,214,133,212,101
6210 DATA 205,133,214,165,215,133,213
6220 DATA 165,0,133,215,164,203,165
6230 DATA 206,240,10,177,214,209,212
6240 DATA 144,44,240,12,176,19,177
6250 DATA 214,209,212,144,13,240,2
6260 DATA 176,38,200,196,204,240,227
6270 DATA 176,23,144,223,169,1,133
6280 DATA 218,164,205,136,177,214,72
6290 DATA 177,212,145,214,104,145,212
6300 DATA 192,0,208,241,232,224,0
6310 DATA 208,2,230,207,228,208,208
6320 DATA 172,165,209,197,207,208,166
6330 DATA 165,218,201,0,208,144,96
7000 REM
            CIO TO GET BYTES
7010 TRAP 7100
7020 IOCB=832+IO*16:POKE IOCB+2,7
7030 ADRHI=INT(ADDRESS/256)
7040 ADRLO=ADDRESS-ADRHI*256
7050 POKE IOCB+4,ADRLO:POKE IOCB+5,ADR
```

SORT BASIC CON'T

```
MI
7060 NUMMI=INT(NUMBER/256)
7070 NUMLO=NUMBER-256*NUMMI
7080 POKE IOCB+8,NUMLO:POKE IOCB+9,NUM
MI
7090 I=USR(ADR("hhh▒LV@"),IO*16)
7100 TRAP 44444:CLOSE #IO:RETURN
```

C BY WALDEN

```
/**** STUDY.C ***/

/* A normal for( loop */
  for(i=1;i(10;++i)

/* A complex for( loop */
  for(k=i=0,j=3; val)4; k+=12)

/* An infinite for( loop. You could
/* get out of this with the break;
/* statement
  for(i=0;;++i)

/* An example of the ?: statement
  z= (a)b)? a:b
/* The above will assign to z either
/* a or b depending on which is larger


main() $( /* run this program!
  int i;
  for(i=1;i(3;++i) $(
/* DVC recognizes only one ?: per line

   printf("There %s",
      (i)1)? "are" : "is");
/* so we split the printf statement
    printf(" %d player%s\n",i,
      (i)1)? "s." : ".");
  $)
$)

main() $( /* run this program!
  int i;
/* this will count the number of keys
/* pressed until the BREAK key
  for(i=0;getchar())=0;++i);
  printf("\n%d key%s",i,
     (i == 1)? " " : "s ");
  printf("%s pressed.\n",
     (i == 1)? "was" : "were");
$)
/* NOTE: == is used instead of = in C
/* when used to compare two values
/* rather than assign one value to
/* another.  (i == 1) NOT (i=1)!
/* It's a common mistake to make if
/* you're used to programming in BASIC
```

```
/*  MENU.C          Written in C   */
/*  This program lists a menu of   */
/*  ".COM" files, and allows the   */
/*  user to select which file      */
/*  to run.                        */

main() $(
  int iocb,place,count,i,j,k,lfmargin;

  char dirname[20],tmp[20],
filename[17*64];

/* A wildcard will be used unless  */
/* the user specifies otherwise.   */
/* The default drive will be added */
/* by the normalize() function.    */
  if(!getdos(dirname)) strcpy(dirname,

"*");

/* don't allow any extentions */
  if((i=strchr(dirname,'.')) ) 0)
dirname[i]=0;
  normalize(dirname,"COM");
  if((iocb=copen(dirname,'d')) ( 0) $(

    printf("Can't open directory of
%s\n",dirname);
/* fatal error, so return to DOS */
    exit();
  $)
  count=0;

 /* read all the matching filenames */

  while(cgets(tmp,iocb) ) 0) $(
      /* start with the drive */
    strcpy(filename+count,dirname);
       /* add the primary name */
    for(place=3,i=2;i(=10;++i) $(
       /* stop if it's a space */
      if(isspace(tmp[i])) break;
      else filename[count+(place++)]=
tmp[i];
    $)
      /* end of the string */
    filename[count+place]=0;
      /* add the extension */
    strcat(filename+count,".COM");

/* Use the following line as a templat
e
/* to remove any files you don't want
/* to include in the menu */

    if(index(filename+count,":CC."))
 continue;

    else count+=17; /* next filename *
/
  $)
 /* remove the FREE SECTORS filename *
/
  count-=17;
  closeall(); /* close the directory *
/
  if(!count) $(
    printf("NO FILES!\n");
 /* Can't do a menu with no files! */
    exit();
  $)

/* Now we print the menu */

  putchar('\f');/* clear the screen */

    /* for future use */
  lfmargin=peek(0x52);
    /* Print all filenames */
  for(place=i=0;i(26 && place(count;
++i,place+=17) $(
    printf("%c- ",'A'+i);
     /* print the primary name only */

    for(k=3;filename[place+k] != '.';
++k) putchar(filename[place+k]);
    putchar('\n');/* add the return */

    if(i == 14) $( /* Second column */

       /* set the left margin */
      poke(0x52,20);
 /* reposition the cursor at the top o
f
 /* the next column */
      position(20,0);
    $)
  $)
    /* restore the left margin */
  poke(0x52,lfmargin);
  position(2,23);
  printf("SELECTION?");
 /* until we get the correct input */
  while(isalpha(i=toupper(getkey()))
&& (i-'A' ) count/17));
      /* go run it! */
    chain(filename+((i-'A')*17));
$)
```

SORT ACTION

```
;       FILE: SORTNUM.ACT


;*************************************
*
MODULE          ;DEFINE GLOBAL VARIABLE
S
;*************************************
*


BYTE KEY,FLAG=[0],TYP,HFLAG

CARD ARRAY S(1000)

CARD   N,K,I,T,M,
       R,N2,L,LL,HOLD,RR,J,HOLD2,JJ,R2
;*************************************
*
PROC BUBBLE()           ;BUBBLESORT
;*************************************
*
K=N

DO              ;TESTPRT()
T=0

    FOR I=2 TO K
    DO
       IF S(I-1)>S(I) THEN
       T=I:M=S(I-1):S(I-1)=S(I):S(I)=M


       FI
    OD

;   IF I=10*(I/10) THEN  PRINT(".") FI

K=T-1
UNTIL T=0
OD

PRINTE(" ")
RETURN
;*************************************
*
PROC HEAPSWAP()     ;FOR HEAPSORT
;*************************************
*

J=LL

DO
   I=J
   HFLAG=1
   J=2*J
   JJ=J
   RR=R

       IF JJ<RR THEN
              IF S(J)<S(J+1) THEN
              J=J+1    FI
       FI


       IF JJ<=RR THEN
              IF HOLD<S(J) THEN
              S(I)=S(J)  HFLAG=0 FI
       FI


       IF JJ>RR OR HFLAG=1 THEN
              HOLD2=HOLD:HOLD=S(I)
              S(I)=HOLD2
       FI


UNTIL HFLAG=1

OD

RETURN
;*************************************
*
PROC HEAP()     ;HEAPSORT
;*************************************
*
R=N     N2=N/2
FOR L=1 TO N2
DO
       LL=N2+1-L        ;NO STEP -1!
       HOLD=S(LL)
       HEAPSWAP()
;      IF L=10*(L/10) THEN  PRINT(".")
 FI
OD     PRINTE(" ")
LL=1
FOR R2=1 TO N-1
DO
       R=N-R2           ;NO STEP -1!
       HOLD=S(R+1)
       S(R+1)=S(1)
       HEAPSWAP()
;      IF R=10*(R/10) THEN  PRINT(".")
 FI
OD     PRINTE(" ")

RETURN
;*************************************
*
PROC GETKEY()           ;GETKEY ROUTINE

;*************************************
*


POKE(764,255)

       DO UNTIL PEEK(764)#255 OD

       KEY=PEEK(764)
       POKE(764,255)


       RETURN
;*************************************
*
PROC TIMER()    ;TIMER ROUTINE
;*************************************
*
       CARD time
       BYTE p18=18,p19=19,p20=20

       flag = 1-flag
       IF flag=0       ; not 1st call
       THEN    time = p20 + 256*
                     (p19 + 256*p18)
               Print("Elapsed Time: ")

               PrintC(time/60)
               PRINT(" AND ")
               PRINTC(TIME MOD 60)
               PrintE("/60 second5")
       FI

       p18=0   ; reset
       p19=0   ; the
       p20=0   ; clock

RETURN
;*************************************
*
PROC FIRSTSCREEN()      ;CLEAR AND PRIN
T
;*************************************
*

BYTE J,FIRST=[0]

IF FIRST=0 THEN GRAPHICS(0) FI
FIRST==+1
POKE(712,0)
J=RAND(16)    POKE(710,4+16*J)
POKE(709,12)

PRINTE(" ")
PRINTE(" Which sort to execute?")
PRINTE(" ")
PRINTE(" (B) Bubblesort")
PRINTE(" ")
PRINTE(" (H) Heapsort")
PRINTE(" ")
PRINTE(" ")

RETURN
;*************************************
*
```

RESCUE

```
8 POKE 710,0:? CHR$(125):? :? "  PLEASE WAIT, CHECKING DATA!":FOR DEL=0 TO 10:NEXT DEL
10 DATA 0,1,2,3,4,5,6,7,8,9,0,0,0,0,0,0,0,10,11,12,13,14,15
15 OPEN #1,8,0,"D:RESCUE.DAT"
20 DIM HEX$(91),Y$(1),HEX(22):FOR X=0 TO 22:READ N:HEX(X)=N:NEXT X:LINE=990:AD=20736:MEM=AD:TRAP 60
25 TOTAL=0:LINE=LINE+10:? "   LINE:";LINE:READ HEX$:IF LEN(HEX$)<>90 THEN 110
28 DATLIN=PEEK(183)+256*PEEK(184):IF DATLIN<>LINE THEN POKE 559,34:? "LINE ";LINE;" MISSING!!!":END
30 FOR X=1 TO 89 STEP 2:D1=ASC(HEX$(X,X))-48:D2=ASC(HEX$(X+1,X+1))-48:BYTE=HEX(D1)*16+HEX(D2)
40 PUT #1,BYTE:TOTAL=TOTAL+BYTE:NEXT X:READ CHK:IF TOTAL=CHK THEN POKE 53279,0:GOTO 25
50 GOTO 120
60 IF PEEK(195)<>6 THEN 130
70 POKE 559,34:? CHR$(125);CHR$(253);" WARNING, MAKE SURE PROGRAM IS SAVED!!"
75 ? :? :? "  READY TO RUN(Y/N)";:INPUT Y$
80 TRAP 100:CLOSE #1:IF Y$="Y" THEN OPEN #1,4,0,"D:RESCUE.DAT"
85 IF Y$<>"Y" THEN END
90 GET #1,BYTE:POKE MEM,BYTE:MEM=MEM+1
95 GOTO 90
100 IF PEEK(195)=136 THEN A=USR(AD):END
105 POKE 559,34:? "ERROR ";PEEK(195):END
110 POKE 559,34:? "NUMBER MISSING OR ADDED!":END
120 POKE 559,34:? "BAD DATA!":END
130 POKE 559,34:? "ERROR ";PEEK(195):END
1000 DATA 2065E4A05FA256A907205CE4A9A68D0002A9578D0102A9C08D0ED4A9CA8DC002A95E8D98598D00D0A99A8D9959,5404
1010 DATA A9688D07D4A93E8D2F02A9038D1DD0A9028D6F02A9D68D3002A9598D3102A9E08DF402200053A9148DB359A901,4844
1020 DATA 8DC059A9008DC6028D03D2AD1FD02901D00AAD1FD02901F0F94C7E51AD8402D0EA4C7E51A910A2069D3D58E8E0,5331
1030 DATA 0AD0F8A9108D5758A9118DBC598D5858A9028DBA59A90B8D50588D4F588D4E588DC102A9008D3002A9588D3102,4724
1040 DATA A9F88DC3598DC659A9028DB859A9038DBF59A9FC8DC859A9008DC9598DB959A9508DB6598DB459200053A9008D,5938
1050 DATA C0598D04D485CB206352A9688DF402A9948DC60220AF52A99A8D9959A9BE8DB5598D04D0A2CFA9019D006BE8E0,5882
1060 DATA D4D0F8A9048DB959A9008DC1598DC7598DC5598DBD598D1ED08DC4598DB7598D9A598D9C598DCD598DBE598DCA,6309
1070 DATA 59AA208D5220FC55A9008DC4598DC5598D1ED0A9248D01D2A98E8D00D2A95E8D98598D00D04C2D569D0E58E8E0,5523
1080 DATA E8E027D0F660EEBA59A9148DBD59200A56A9AA8D01D28D00D220FC55A9788D00D220FC55A9008D01D28DC45920,5685
1090 DATA 805320A55520FC55CEC859CEB459ADC65938E9058DC6594CDE51A200BDA1589D9F59E8E014D0F560A200ADC159,6040
1100 DATA DDC258F006E8E00ED0F3608DB558E8E8E89D9F59CABDA1589D9F5960CE9D59F00160AD78028D9E592901D003CE,6397
1110 DATA 9959AD9E592902D003EE9959AD9E592908D008EE9859A9008D9C59AD9E592904D008CE9859A9048D9C59AD9859,5216
1120 DATA C99BD008CE9859A9018DC259AD9859C92FD003EE9859AD9959C949D003EE9959A90F8D9D59AD9A59C904D005A9,5919
1130 DATA 008D9A59AC9A59B94959186D9C59AAEE9A59A900186908CAD0FAA8A9008D9B59AE9959B9D1589D006CE8C8CE9B,5827
1140 DATA 59D0F3AD98598D00D060A900AA9D006B9D006C9D006D9D006E9D006FCAD0EE60ADC159C9419006ADCD59D05B60,5545
1150 DATA CECC59ADCC59F00160A9548DCC59ADCD59D047EEBD59AD0AD28D1ED0C94A90F6C99BB0F2A8A9008DD1598CCE59,6673
1160 DATA A2A38E02D08ECF59A9008D9B59A2408ED059BDD15899006EE8C8CE9B59AD9B59D0F0ADD159D0C2A9018DCD5960,6412
1170 DATA ADCB59D015ADD159D008A9408DCA594C1754ACCE59A9004C5154A9018DCB59ADD0591869008DD059AAC970900E,5395
1180 DATA A9008DCD598DCB59AD0AD28DCC59ACCE59A9008D9B59BDD15899006EC8E8CE9B59AD9B59D0F06099006EC8C008,6148
1190 DATA D0F84CB953ADC159C9A0B00160CED259F00160A9018DD259ADD359D036ADC159C9CB900160A9018DD359EEBD59,6303
1200 DATA A9D88DD4598D01D0AD0AD2C94990F9C968B0F5A88CD559A200BD4E5999006DC8E8E04AD0F460CED459ADD4598D,6753
1210 DATA 01D0C91E900160A9008DD359A900AAACD55999006DE8C8E04AD0F760CEC359ADC359F00160CEB859F00160ADC6,6118
1220 DATA 598DC359A9028DB859CEB559ADB5598D04D0C95EF00160A9008D01D28D01D08D02D08D06D2EE9959A9108D9B59,5632
1230 DATA 205456CE9B59AD9B59D0F5A900A201205D53ADC559F0034C2D55ADC459F0094C6E52A9008D01D28D07D2200053,5275
1240 DATA A900ACCE59A20099006EC8E8E000D0F78DC159205456A2408ED0598ECA59AC9959A9008D9B59AED059BDD15899,5989
1250 DATA 006CE8C8CE9B59AD9B59D0F0A9358D9B59CE9B59205456AD9B59D0F5205456ADD0591869008DD059C97090C5A9,6062
1260 DATA 008DC5598D1ED020D4552054562054564CDE51ADBA598DBB59A9108D58588D5758A01BCEBB59D00160B93D58C9,5110
1270 DATA 1AF009186901993D584CB355A910993D58884CBB55CEBF5920FC558D4E588D4F588D5058ADBF59F00FACBF59A2,5119
1280 DATA 13A90B9D3D58CA88D0F96068684C4151A9008D9B59205456CE9B59D0F860EEBD59A209CEBD59D00160BD3D58C9,5599
1290 DATA 19F00869019D3D584C8D56A9109D3D58CA4C155620DE52209653205456 20CD54200A56ADC459F0034C6E52ADC5,4327
1300 DATA 59F0034C2D55ADBE59D0FB4C2D56A200A003CAD0FD88D0FA60ADC059F015CEB359D000A9208DB359ADF3024902,5847
1310 DATA 8DF3024C62E4ADFC02C921D008A9FF8DFC02ADBE5949FF8DBE59ADBE59F0034C62E4EEC202205C54ADB759D00F,6096
1320 DATA ADC159C991D008A90F8DBD598DB759AD06D02903F003EED159AD04D02908F003EEC559AD04D02904F003EEC459,5643
1330 DATA AD0CD02906F003EEC559ADCA59F00D4A4A8D03D2A9288D02D2CECA59ADC259F041CECF59ADCF598D02D0CED459,6075
1340 DATA ADD4598D01D0A9008DC259C6CBA5CB8D04D41004A90785CB8D04D4854DEE0E58AD0E58A2009D0E58E8E8E8E027,5765
1350 DATA D0F6EEC159208D52AD0AD2CDC8599029ADC75949CF8DC759ADC759A00099E45FC8C009D0F8A00099E461C8C009,6512
1360 DATA D0F8A00099E463C8C009D0F8CEB459F0034CA357ADB6598DB459ADB9590AAAA9009D7D5E9D7D5F9D8E659D8E66,6281
1370 DATA CEB959ADB959D005A9048DB959ADB9590AAAA97D9D7D5E9D8E65A97C9D7D5F9D8E664C62E4488A48A200BDC057,5810
1380 DATA 9D16D0E8E003D0F5AD0AD28D19D068AA68403478C6000000000000000000000000000000000000000000000000,2878
1390 DATA 000000000000000000000000000000000000000000000000000000000000000000000000707070463D582086,721
1400 DATA 30479F593056005B56005C56005D56005E56005F560060560061560062560063560064560065560066560067 46,2864
1410 DATA 655807300641005841424344458A10111516100000000000000000000C8C5C9C5C88A1011000000000000000000,2134
1420 DATA 0000000C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0086,614
1430 DATA 8785882B2C6C6D6C6C6D6C6C6D6C6C6D0000000000002E2F30313233343536373839000000006E6F7071727374,3147
1440 DATA 757677777879000A1E32465A6E82
```

```
96AABED2DFE9FF001F044E3F1F8E00000E044E
3F1F0E000004044E3F1F0E00,3254
1450 DATA 000E044E3F1F0E00007C10397E7C
3800003810397E7C3800001010397E7C380000
3810397E7C3800000000181C,2032
1460 DATA 0000000012205C380A2000104410
5A0D2204000410249908224004896088A10948
24C1240048010000042000000,1785
1470 DATA 0000000000000001020304010307
3F7EFE7E3F07030100000000000000000000000
0000000000000000000010307,675
1480 DATA 3F7EFE7E3F07030100000000000
0000000000000000000000000000000103073F7E
FE7E3F07030100SEA1010000,1553
1490 DATA 010F0000006E2F303132333435363
7373839000000001E475E5000020402001100
0000010000F80000F800FC00,1002
1500 DATA 0000680089A37888010054667070
7070707047F45970700770707070067070067070
70700270707041D659000000,3744
1510 DATA 72006500730063007500650000000
0000000000000006D0065007300730069006F
006E0000000223900272SF,1631
1520 DATA 26260034282F2D30332F2E000026
2F320030212725001600202127213A292E2500
0000000000000000000000,1098
1530 DATA 0000303225333300B3B4A1B2B400
0000000000000000000000000000000000000000
0000000000000000000000,1115
1540 DATA 0000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000,0
1550 DATA 0000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000,0
1560 DATA 0000000000000000000000000000
0000000000000000000000000000000000000000
00000000000000C0C000000,24
1570 DATA 0000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000,0
1580 DATA 0000000000000000000000000000
0000000000000000000000000000000220C0C
0C0C0C0C0C0C0C0C0C0C0C,202
1590 DATA 0C0C0C0C0C0C0C0C0C0C0C0C0C0C
0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C
0C0C0C0C0C0C0C0C0C0C0C,540
1600 DATA 0C0C0C0C0C0C0C0C0C0C0C0C0C0C
0C0C0C0C0C0C0C0000000000000000000000000
0000000000000000000000,252
1610 DATA 0000000000000000000000000000
0000000000000000000000000000000000000000
00000000000000C0C0C0C0C,60
1620 DATA 0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C
0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0000000000
0000000000000000000000,336
1630 DATA 0000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000,0
1640 DATA 0000000000000000000000000000
000000220C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C
0C0C0C0C0C0C0C0C0C0C0C,358
1650 DATA 0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C
0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C
0C0C0C0C0C0C0C0C0C0C0C,540
1660 DATA 0C0C0C0C0C0C0C0000000000000000
0000000000000000000000000000000000000000
0000000000000000000000,84
1670 DATA 0000000000000000000000000000
0000000000000000000000000C0C0C0C0C0C0C
0C0C0C0C0C0C0C0C0C0C0C,228
1680 DATA 0C0C0C0C0C0C0C0C0C0C0C0C0C0C
0000000000000000000000000000000000000000
0000000000000000000000,168
1690 DATA 0000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000,0
1700 DATA 00000000D00220C0C0C0C0C0C0C
0C0C0C230D22230000D220C0C0C0C0C0C0C0C0C
0C0C0C0C0C0C0C0C0C0C0C,571
1710 DATA 0C0C0C0C0C0C0C0C0C0C0C0C0C0C
0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C
0C0C0000000000000000000,420
1720 DATA 0000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000,0
1730 DATA 0000000000000000000000000C0C
0000000000000000000000000000000C0C0C0C
0C0C0C0C0C0C0C0C0C0C0C,216
1740 DATA 0000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000,0
1750 DATA 0000000000000000000000000000
0000000000000000000000000000000000000000
0000000000220C0C0C0C0C0C,186
1760 DATA 0C230000000000000000220C0C
0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C
230000000000007D000022,527
1770 DATA 0C0C0C0C0C0C0C0C0C0C0C0C0C0C
0C0C0C0C0C0C0C000C0C0000000000000000000
0000000000000000000000,276
1780 DATA 0000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000C0C,24
1790 DATA 0000000000000000000000000000
AA0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0000
0000000000000000000000,362
1800 DATA 0000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000,0
1810 DATA 0000000000000000000000000000
0000000000000000000000D2928272300D0000
0000000000001B1A0000220C,280
1820 DATA 0C0C0C0C0C0C0C0C0C0C0C0C0C0C
0C0C0C0C230000000000000007C000000220C
0C0C0C0C0C0C0C0C0C0C0C0C,565
1830 DATA 0C0C0C0C0C230000C0C0000000
0000000000000000000000000000000000000000
0000000000000000000000,0
1840 DATA 0000000000000000000000000000
0000000000000000000000000000000000C0C7A00
0000000000000007B0C0C0C,305
1850 DATA 0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C
0C0C0C1A0000000000000000000000000000000
0000000000004E0000000000,388
1860 DATA 0000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000,0
1870 DATA 00004E0000000000000000000000
0000000000000001B0C0C0C0C0000D0022
0C0C0C0C0C0C0C0C0C0C0C0C,356
1880 DATA 0C0C0C230000B0C0C0C0C0C0C0C
0C1A000000C0C0C0C0C0C0C0C0C0C0C0C0C0C00
00000001B0C0C0C00000000,451
1890 DATA 0000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000,0
1900 DATA 0000000000000000000000000000
000C0C0000000000000000000000C0C0C0C0C0C
0C0C0C0C0C0C0C0C0C0C0C,228
1910 DATA 0C0C0C0C1A0000000000000000000
0000000000000000001B0C1A0000000000000
0000000000000000000000,139
1920 DATA 0000000000000000000000000000
0000000000000000000000000000000000001B
0C1A00001B0C1C1D1E00001B,218
1930 DATA 1A0000000000001B0C0C0C0C0C0C
1A0000000220C0C0C0C0C230D00220C0C0C0C
230000B0C0C0C0C0C0C0C,533
1940 DATA 0C0C00000C0C0C0C0C0C0C0C0C0C
0C0C0C0C00000C0C0C0C0C0C0C0000000000000
0000000000000000000000,276
1950 DATA 0000000000000000000000000000
0000000000000000000000000000000000000000
0000000000000000000000,0
1960 DATA 000C0C7A0000000000000000007B
0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C
0C0C0C1A0000000000000000,559
1970 DATA 0000000000000B0C0C0C0C0C1A
0000000000000000000000000000000000000000
0000000000000000000000,113
1980 DATA 4E0000000000000000000000000
0021201F0C0C0C1A1B0C0C0C0C001B0C0C1A
00000001B0C0C0C0C0C0C0C,511
1990 DATA 0C0C0C0C1A00220C242526000000
0000220C230000B0C0C0C0C0C0C0C0C0C0C0C
00000C0C0C0C0C0C0C0C0C0C,591
2000 DATA 0C0C0C0C00000C0C0C0C0C0C0C00
0000000000000000000000000000000000000000
0000000000000000000000,132
2010 DATA 0000000000000000000000000000
0000000000000000000000000000000000000000C
0C0000000000000000000000,24
2020 DATA 0C0C0C0C0C0C0C0C0C0C0C0C0C0C
0C0C0C0C0C0C0C0C0C0C1A0000000000000000000
0000004E1B0C0C0C0C0C0C0C,491
2030 DATA 1C1D1E0000000000000000000000
```

RESCUE CON'T

```
0000000000000000000000000021201F0C0C1A
00000000000000000004E1B,338
2040 DATA 0C0C0C0C0C0C0C0C0C0C0C0C0C0C
2230 DATA 0C0C000000000000000000000000
0000000000000000000000000000000000000
00000000000000000000,24
2240 DATA 0000000000000000000000000000
0000000000000000000000000000000000000
000C0C0C0C0C0C0C0C0C0C0C,132
2250 DATA 0C0C0C0C0C0C0C0C0C0C0C0C0C0C
0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C
0C0C0C0C0C0C0C0C0C0C0C0C,540
2260 DATA 0C0C0C0C0C0C0C0C0C0C0C0C0C0C
0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C
0C0C0C0C0C0C0C0C0C0C0C0C,540
2270 DATA 0C0C0C0C0C0C0C0C0C0C0C0C0C0C
0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C
0C0C0C0C0C0C0C0C0C0C0C0C,540
2280 DATA 0C0C0C0C0C0C0C0C0C0C0C0C0C0C
0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C0C
000000000000000000000000,396
2290 DATA 0000000000000000000000000000
0000000000000000000000000000000000000
000000000000000000000000,0
2300 DATA 0000000000000000000000000000
000000000000C0C0C0C0C0C0C0C0C0C0C0C0C
0C0C0C0C0C0000000000000000,216
2310 DATA 0000C603C0E0E7E3C003C666066
7E7E3C003C6666667E7E3C007C66667C7E6E66
007E6078607E7E7E007E607C,3732
2320 DATA 70707070006666666667E7E7E0060
6060607E7E7E006666666667E3C180018180018
181800001F044E3F1F0E00FF,3210
2330 DATA FFFFFFFFFFFFFF7F7F3E3E1C1C08
0800101010387CFEFE000000FF0000000003C
6E76667E7E3C00183818187E,4182
2340 DATA 7E7E003C660C187E7E7E007E180C
0C667E3C000C1C3C6C7E7E0C007E607C06667E
3C003C607C667E7E3C007E06,3228
2350 DATA 0C1838383800 3C663C667E7E3C00
3C663E060E3C3800C0E0F0F8FCFEFF0103070F
1F3F7FFFE0FCFFFFFFFFFFFF,5501
2360 DATA 000080F0FEFFFFFF0000000000C0
F8FF073FFFFFFFFFFFFF0000010F7FFFFFFF00
000000031FFFFF7F3F1F0F,5370
2370 DATA 070301FFFEFCF8F0E0C080FFFFFF
FFFFFFFCE0FFFFFFFEF0800000FFF8C0000000
0000FFFFFFFFFFFF3F07FFFF,8002
2380 DATA FF7F0F010000FF1F0300000000000
000000000000FFFF0000000000010101000000
00000011FF00000000000111,1490
2390 DATA FFC0C0C0C0C1F3FFFF00000060F0
F8FDFF000000000377FFFF0000000287CFFFFF
FF3E1041EBFFFFFFFF7F3D98,6540
2400 DATA C0E7FFFFFFE3C19D3DFDFCFFFFFC
F9FBFB8307FF0000800800800FF0000000000
0000FF7F404141414101FFFF,6477
2410 DATA 3FFFFFFFFFFFFC0F0FCFFFCF0C0
000300F3FFF3F0F0300FF7E7E3C3C1818000018
183C3C7E7EFF000000000000,4983
```

ANALOG INPUT DEMO FROM AUGUST

```
10 REM ANALOG INPUT DEMO
15 OPEN #2,4,0,"K:
20 GR.2+16
30 ?#6;"ANALOG INPUT DEMO
40 ?#6;"1.TEMPERATURE
50 ?#6;"2.LIGHT LEVEL
60 ?#6;"3.HUMIDITY
62 ?#6;:?#6;"PLUG SENSOR INTO PINS 7 A
ND 9 OF JACK 1
75 GET#2,A
80 ?#6;A-48
90 FOR TIME=1 TO 80:NEXT TIME
95 ON A-48 GOTO 100,200,300
98 G.999
100 GR.2+16
110 POKE 752,1
120 P=PEEK(624)
125 POS.0,0
130 T=(38-P)*18/13+74
140 ?#6;"THE TEMPERATURE IS
150 ?#6;INT(T);" DEGREES
160 IF PEEK(764)<>255 THEN GET #2,A:G.
20
170 G.120

200 GR.2+16
210 POKE 752,1
220 P=PEEK(624)
225 POS.0,0
230 ?#6;"THE LIGHT LEVEL IS
235 L=100-P*100/228
240 ?#6;INT(L);"    "
250 IF PEEK(764)<>255 THEN GET #2,A:G.
20
260 G.120
270 GR.2+16
280 POKE 752,1
290 P=PEEK(624)
300 GR.2+16
310 POKE 752,1
320 P=PEEK(624)
325 POS.0,0
330 ?#6;"THE HUMIDITY IS
335 H=100-P*100/228
340 ?#6;INT(H);"    "
350 IF PEEK(764)<>255 THEN GET #2,A:G.
20
360 G.120
999 GR.1+16
1000 ?#6;"PARTS LIST";?#6;:?#6;"TEMPER
ATURE SENSOR: FENWAL      THERMISTOR G
A45P2,50000 OHMS
1010 ?#6;"AT 25 DEGREES C.
1020 ?#6;:?#6;"LIGHT SENSOR:
    GENERAL ELECTRIC X-6 PHOTOCELL
1030 ?#6;:?#6;"HUMIDITY SENSOR:
    DEVRY INDUSTRIES HYGROPAK MODEL HA

1040 G.1040
```

SORT ACTION

```
PROC INIT()     ;INITIALIZATION ROUTINE
;*************************************
*

CARD A

DO
       FIRSTSCREEN()
       GETKEY()
       IF KEY=21 THEN TYP='B  FI
       IF KEY=57 THEN TYP='H  FI

UNTIL TYP='B OR TYP='H
OD

DO
       PRINT("How many items do ")
       PRINT("you want to sort")
       PRINTE(" ")
       PRINTE(" ")
       PRINTE("Max = 999, Minimum = 10
")
       PRINTE(" ")
       PUT('?)  N=INPUTC()
UNTIL N)9 AND N<1000
OD


OPEN(3,"D:NUM1000.RAN",4,0)
;PEN(3,"D:NUM1000.SRT",4,0)
FOR I=1 TO N
DO
       A=INPUTCD(3)
       S(I)=A
OD
CLOSE(3)


TIMER()
RETURN



;*************************************
*
PROC PRNTS()          ;PRINT RESULTS
;*************************************
*

CARD TEMP

GETKEY()

FOR I=1 TO N
DO
IF KEY=28 THEN EXIT FI
       PRINTC(I)
       PRINT(")")
       TEMP=S(I)
       PRINTCE(TEMP)
OD

RETURN




;*************************************
*
PROC MAIN()
;*************************************
*

DO
INIT()

IF TYP='B  THEN BUBBLE() FI
IF TYP='H  THEN HEAP()   FI

TIMER()
POKE(752,1)
PRINTE(" ")
PRINTE("   Press any key to see list")

printe("       Press ESC to exit")
PRNTS()
IF KEY=28 THEN EXIT FI



OD

RETURN
```

PROOFER IN LOGO

```
TO MAIN.PROC
MAKE "LIST RL
IF ( FIRST :LIST ) = "TO [MAKE "TITLE
FIRST BF :LIST MAKE "INPUTS BF BF :LIS
T]
IF (FIRST :LIST ) = "END [FINISH.UP ST
OP]
IF NOT (FIRST :LIST ) = "TO [MAKE "INS
TRUCTIONS LPUT :LIST :INSTRUCTIONS]
MAKE "SUMSLIST SE :SUMSLIST ADDSUMS CH
KLINE :LIST
TYPE []] MAIN.PROC
END

TO CHKLINE :LIST
IF EMPTYP :LIST [OP []]
IF LISTP FIRST :LIST [OP SE CHKLINE FI
RST :LIST CHKLINE BF :LIST]
OP SE CHKWRD FIRST :LIST CHKLINE BF :L
IST
END

TO DIGIT3 :NUMBER
IF ( COUNT :NUMBER ) < 4 [OP :NUMBER]
OP ( WORD LAST BL BL :NUMBER LAST BL :
NUMBER LAST :NUMBER )
END

TO CHKSUM :NUMBER :COUNT
OP :COUNT * :NUMBER
END

TO CHKWRD :WORD
IF EMPTYP :WORD [OP 0]
OP DIGIT3 SUM CHKSUM ASCII LAST :WORD
COUNT :WORD CHKWRD BL :WORD
END

TO ADDSUMS :LIST
IF EMPTYP :LIST [OP 0]
OP DIGIT3 SUM CHKSUM LAST :LIST COUNT
:LIST ADDSUMS BL :LIST
END

TO FINISH.UP
DEFINE :TITLE LIST :INPUTS :INSTRUCTIO
NS
MAKE THING "TITLE LIST ADDSUMS :SUMSLI
ST :SUMSLIST
CT TYPE [Any more procedures to enter?
]
MAKE "LIST RC PR :LIST
IF :LIST = "Y [LOGO.PROOF]
PR [] TYPE [Output to Printer?]
MAKE "LIST RC PR :LIST
IF :LIST = "Y [COPYON]
ERN [TITLE LIST INSTRUCTIONS]
ERN [INPUTS SUMSLIST]
PR [] PONS PR [] COPYOFF
TYPE [Do you want to quit now?]
MAKE "LIST RC PR :LIST
IF :LIST = "N [LOGO.PROOF]
ER [CHKWRD CHKSUM DIGIT3]
ER [CHKLINE ADDSUMS MAIN.PROC]
ER [LOGO.PROOF FINISH.UP]
ERNS RECYCLE
END

TO LOGO.PROOF
MAKE "INSTRUCTIONS []
MAKE "SUMSLIST []
CT PR [Enter your procedure:] PR []
```

PROOFER CON'T

```
TYPE [?] MAIN.PROC
END

TO START
CT REPEAT 15 [PR []]
PR [|THE LOGO PROOFREADER|]
PR [] PR []
PR [By Dave Arlington]
PR [for ATARI Explorer]
REPEAT 10 [PR []]
PR [##(Press any key to continue.#)]
MAKE "LIST RC ER "START RECYCLE
LOGO.PROOF
END

TO COPYON
SETWRITE "P:
END

TO COPYOFF
SETWRITE []
END
```

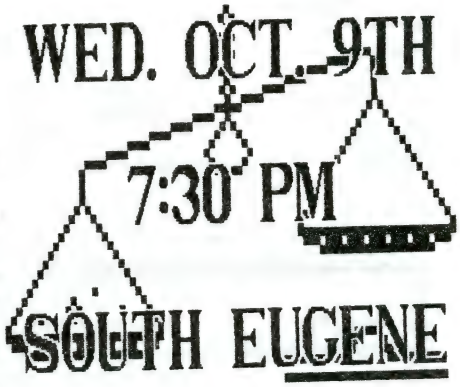
SWITCH 2.5

```
10 REM Modification to DOS 2.5 to
11 REM store DUP.SYS and MEM.SAV
12 REM in the bank switch RAM
13 REM behind the OS ROM from $C000
14 REM to $F8ff
15 REM
16 REM This mod for 64K XL's only
20 REM Adapted from ANALOG #24 by
21 REM Robert Luce
22 REM
23 REM ******************************
24 REM written by Alec Benson 6/85
30 REM from FEEDBACK ADELAIDE Atari
31 REM Club, Box 333, Norwood,
33 REM Australia S.A. 5067 Aug '85
34 REM ******************************
40 REM REPRINTED ACE Newsletter
41 REM 3662 Vine Maple, Eugene, OR
100 CK=0:DIM A$(339)
105 ? :? "Reading Data...."
110 FOR I=1 TO 339
120 READ A
130 CK=CK+A
140 A$(LEN(A$)+1)=CHR$(A)
150 NEXT I
160 IF CK<>41072 THEN ? "ERROR IN DATA
 STATEMENTS-CHECK TYPING":END
170 OPEN #1,8,0,"D:PATCH25.OBJ":PRINT
#1;A$;:CLOSE #1
180 ? :? :? "D:PATCH25.OBJ CREATED:END
"
1000 DATA 255,255,231,28,233,28,32,192
1010 DATA 23,70,23,138,23,32,85,24
1020 DATA 169,0,133,212,133,214,169,29

1030 DATA 133,215,169,192,133,213,162,
16
1040 DATA 32,119,24,169,216,133,213,16
2
1050 DATA 7,32,119,24,32,70,24,96
1060 DATA 169,0,133,212,169,224,133,21
3
1070 DATA 160,0,162,3,177,212,72,32
1080 DATA 85,24,104,145,212,32,70,24
1090 DATA 200,208,241,230,213,202,16,2
36
1100 DATA 96,234,162,23,0,24,240,73
1110 DATA 32,70,23,206,158,23,48,65
1120 DATA 32,108,21,32,105,23,169,255
1130 DATA 141,158,21,141,157,21,162,16

1140 DATA 169,47,157,68,3,169,24,157
1150 DATA 69,3,32,164,21,32,85,24
1160 DATA 162,21,169,0,133,212,133,214

1170 DATA 169,31,133,215,169,228,133,2
13
1180 DATA 32,119,24,32,70,24,169,0
1190 DATA 141,157,21,141,158,21,76,146

1200 DATA 25,19,24,39,24,32,85,24
1210 DATA 169,0,133,214,133,212,169,22
6
1220 DATA 133,215,169,31,133,213,162,2
1
1230 DATA 208,18,58,24,146,24,32,119
1240 DATA 24,32,70,24,206,157,21,76
1250 DATA 152,32,32,102,24,88,169,112
1260 DATA 141,14,212,169,10,141,14,210

1270 DATA 96,120,169,0,141,14,212,141
1280 DATA 14,210,173,1,211,41,254,76
1290 DATA 107,24,173,1,211,9,1,141
1300 DATA 1,211,96,234,234,234,234,32
1310 DATA 156,25,96,160,0,177,214,145
1320 DATA 212,200,208,249,230,213,238,
215
1330 DATA 202,208,242,96,234,234,234,2
34
1340 DATA 234,234,234,234,234,234,234,
63
1350 DATA 25,109,25,32,85,24,169,0
1360 DATA 133,212,133,214,169,29,133,2
13
1370 DATA 169,192,133,215,162,16,32,11
9
1380 DATA 24,169,216,133,215,162,7,32
1390 DATA 119,24,32,70,24,96,234,234
1400 DATA 234,234,234,234,234,234,234,
234
1410 DATA 234,234,49,31,53,31,178,174
1420 DATA 181,216,204
```

FLOATING PT. FROM LAST MONTH: BY BARKLEY

```
10 /*Floating Point Averages*/
20 /*first try with AceC f.p.*/
30 main() ${
40  int i,ave;
50  char *str,a[6],sum[6],n[6],m[6];
60   atof(sum,"0.0");
70    for(i=0;i<19;++i) ${
80     putchar(':');
85     gets(str);
90     atof(a,str);
0100    fadd(a,sum,sum);
0105    printf("     %9.3f   %9.3f\n",
a,sum);
0110   $}
0120   itof(i,n);
0130   fdiv(sum,n,m);
0140   ave=ftoi(m);
0150  printf(" n:   sum:  ave:\n");
0160  printf("%d %5.2f %5.2f\n",i,sum,
m);
0170  getkey();
0180 $}
```

MEETING



HIGH

LOGO PROOFREADER

(Reprint: POKEY, August, 1985)

With the introduction of the amazing new 520ST, Atari has achieved another first. The 520ST is the first inexpensive personal computer to be shipped with LOGO as the language of choice. This is a very exciting and innovative move. In the coming months, many interesting LOGO projects and programs will be published in the Atari Explorer. One side benefit to this is that many of these programs will run with only minor changes on the 400/800/XL/XE series of Atari computers equipped with the LOGO language. The LOGO proofreader is one of these programs.

Almost all computer publications which feature type-in BASIC listings supply a proofreading program which checks your typing. The LOGO Proofreader serves exactly the same purpose for type-in LOGO listings. It provides an easy-to-use checksum for each procedure you enter into the workspace, saving a lot of debugging time. All future LOGO listings published in POKEY will have Proofreader checksums which you can use to check your typing.

Using the Proofreader: Very carefully type the listing. (After all, you don't have the proofreader yet!) When using the 520ST to type in this listing I recommend clicking the LOGO text window to full screen. Then, before doing anything else, save your workspace to disk under the filename LPROOF. Later we'll show you how to use the Proofreader to check itself.

To begin the program, type START. After a short title screen, you will be prompted to being entering a procedure. Just type it in as you normally would. The Proofreader checks each line as you type it. As an example we'll use probably the most famous LOGO procedure, SQUARE.

Type this in:

```
TO SQUARE :SIDE
REPEAT 4 [FD :SIDE RT 90]
END
```

When you type END and press RETURN, the Proofreader will define the SQUARE procedure and ask if you have any more procedures to enter. In this example we'll reply N. After asking whether you also want output to the printer, the LOGO Proofreader will print the checksums for all the procedures you have entered. In this example the Proofreader will print: "MAKE "SQUARE [980 [590 695]]"

The first number in the checksum is ALWAYS the checksum for the entire procedure. In this case, 980. If the number matches, the procedure has been typed correctly and you need check no further.

If, on the other hand, you get something like: "MAKE "SQUARE [970 [590 690]]" then you will have to check the following list of numbers to determine which line was typed incorrectly. In this case, the second line was typed incorrectly. Each number in the list corresponds to each line in the procedure, including the title line but excluding the END line. The LOGO Proofreader assumes you will always type the last line (END) in a procedure definition correctly. It is never included in the checksum values.

After you have finished entering procedures and have determined where all your typing mistakes are, you will have two choices. You will be asked if you wish to quit now. If you answer N, you will be returned to the main program where you may either enter some more procedures or re-type the ones where you have made mistakes. If you answer Y to quitting the LOGO Proofreader will erase itself and all variables including the checksums, leaving only the procedures you have entered within the program. After correcting any mistakes using the LOGO editor, you can then save the entire workspace to disk.

How it Works

If all you are interested in is having the program work, you need read no further. The following information is intended for those who want to know some of the finer points of using list processing in LOGO. One important point which should be made is this: Yes, Virginia, it is possible to write utility and applications programs in LOGO.

Solving a problem in LOGO can be easy if you approach it in the right way. Long before electron microscopes were invented, Greek philosophers discussed the existence of the atom. They theorized that if one took a piece of cloth and kept ripping it in half, that eventually you would get a piece so small that it could not be divided further. The language which LOGO is descended from, LISP, also refers to atoms. Here it is used to describe a piece of a program which cannot be broken down any further. From these small pieces we can construct a larger problem solving program.

LOGO works much the same way. In the Proofreader program, the smallest procedure is CHKSUM, which outputs the ASCII value of a letter times its position in a word. CHKWRD adds these sums together to get a unique value for each word. For instance, CHKWRD "THE outputs 1 times the ASCII value of T plus 2 times the ASCII value of H plus 3 times the ASCII value of E. If we know all the right letters are in the right places, then the word must be spelled correctly.

CHKLINE takes the list the user types in and breaks it up into tits individual words to apss on to CHKWRD. CHKLINE then outputs a list of all the checksums to ADDSUMS. If we know all the words are spelled right and in the right places, then the line must be typed correctly. If all the lines are typed correctly and in the right place, then the procedure must be typed correctly. This is a very good example of solving a problem in LOGO. Take a very small problem (Is the letter in the right place?) and use it to build a foundation to solve a large problem (Is the procedure typed correctly?).

The Proofreader also uses another popular LOGO technique called recursion. a LOGO can call any other procedure, even itself. Recursion is when a LOGO procedure calls itself to help solve a problem. There are generally two types of recursive calles. Tail recursion is where the very last line in a procedure is a call to itself. This is very similar to BASIC looping. The procedure keeps looping back to itself until some condition is met to stop it. In the procedure MAIN.PROC, it keeps going back for more lines from the user until they type END, at which time it calls the FINISH.UP procedure and stops.

A more complicated type of recursion occurs in the procedure CHKLINE. Since CHKWRD can only take a word as its input, CHKLINE must break up a list into its individual words. But a list can contain other lists as well as words. We need a procedure which can break those lists inside lists into words. Sound familiar? That's the CHKLINE procedure! Therefore the second line of CHKLINE checks to see if the first element is yet another list. If it is, it calls itself to break up that list before it continues. This way, CHKLINE can take the most complicated of lists and break it into single words.

The Proofreader uses other techniques which could take up a whole article by themselves. The use of Global vs. Local variables, for instance. The Proofreader uses global variables only in the main loops near top level where user input is needed. All other values are passed through the various procedures locally, where they only exist as long as the procedure is running. This way the program has to only erase a few global variables when it finishes up.

I hope you find the LOGO Proofreader a valuable addition to your library, both as a useful utility and as a lesson in LOGO programming.

WALDEN TEACHES C

In this article I will talk about two C statements which are very powerful, yet difficult to understand, especially if you are used to Basic. Elsewhere in this issue is a program listing called STUDY.C which contains the examples I will refer to.

The Basic FOR statement contains three parts: The initialization where the variable is assigned a value, the test where the variable is checked against some parameter, and the STEP value to increase the variable. The C for(statement also has 3 parts, but unlike Basic, they are totally unrelated to each other.

The variable you initialize does not have have to be either the variable you test or the one you change. Here's how C handles the 3 parts. The first part is used to initalize any number of variables, all separated by commas. The end of the first section is set off by a semi-colon. The variables will all be initialized at the beginning of the loop.

The next section is the test. You may test anything; there is no requirement for the test have anything to do with the initialization. If the test is true, then control will be passed to the body of the loop, otherwise control jumps out of the loop. The third section is performed at the end of the loop. In this section you can assign values, change values, or even leave it blank and do nothing. The variables you change do not have to have anything to do with either of the first two sections. Take a look at the for(examples given in STUDY.C

The second statement for which there is no counterpart in Basic, is the ?: statement. Let's say you wanted to assign Z the value of A or B, whichever is greater. If A is greater than B then Z will equal A, else Z will equal B. Normally this requires two statements. The ?: statement allows you to test a value and give one of two results depending on whether the test value is true or false (I know, it sounds confusing - take a look at the example in STUDY.C). Here's another example. Say you want to print how many players there are in a game. If only one, then the word "players" should be singular, otherwise it should have the ending "s". By using the ?: statement we can print out the ending "s" only if there is more than one player. You could also use the same routine for printing out verbs like "is" and "are" dependent on how many objects you are describing. Look at STUDY.C for examples.

These two statements provide two examples of how C provides shortcuts for commonly used program statements. It takes a while to get used to them, but once you do you may end up wondering how you lived without them.

— Ralph Walden

MENU.C

MENU.C will list a menu of all of the ".COM" programs on a disk, and allow you to run any of them by pressing the corresponding letter. A good use for this program is to end your C programs by chaining to MENU.COM. This way, the user can always have a menu of programs to choose from. The only disadvantage is that you cannot pass parameters to the program being run by MENU.

The listing of the program has been modified somewhat to improve readability. For example, the first char declaration should be typed in as one line rather than the two shown. There are several similar lines which are printed as two lines, but which you can type in as one line. The line which searches for CC.COM by using the index() function can be followed with any number of "else if(" lines to search for and ignore any COM files you don't want listed in the menu.

For those of you who are fairly new to C, I draw your attention to a couple of interesting lines. Look at the for(loop underneath the "Print all filenames" comment. This for(loop initializes, checks, and updates two variables in a single statement. Imagine doing this in Basic! Also look at the while(statement 4 lines from the bottom. With one statement we input a character from the keyboard, change it to uppercase if needed, make sure it's alphabetic and that it's a viable menu option, and keep trying until we get a correct response. This is a good example of C's ability to provide programming "shortcuts"; replacing with one line what normally takes several lines.

— Ralph Walden

A 'C' NOTE

For the past few weeks I've used two new "C"-Language products which look very nice. Although I haven't yet had time to do a complete review, some ACE members might want to look at these new "C" tools for themselves while I do my homework for a review.

Let's look first at "C Self-Study Guide" by Jack Purdum (1985, Que Publishing, Inc., 7999 Knue Rd., Indianapolis, IN, 46250 $16.95). This 250 page soft-bound volume is by far the most cost-effective way I've found for learning to use C. Other C books give you rules, syntax, and finished examples of C programming. With my slow-motion brain, all of this seems to produce one "Gotcha!" after another." What, you didn't notice that comma . . . extra space . . . missing quote? Hah!"

Purdum does it MY way, instead. Lots of examples of C functions, many WITH ERRORS . . . "What's wrong with this line?" (You're told what page holds the answer). That's the way I spend most of my programming time, and the way I really learn — by falling down a lot, and getting up. Not by watching others walk, or run. Purdum also has a super idea which is new to me: He puts the questions for each chapter IN FRONT where you can see how much you already know — or don't know — about Operators, Variables, Loops, Pointers, etc. This makes it simple to skip over familiar ground, and gives me new incentive to identify and fill gaps in my retained knowledge, not merely review the text I just finished reading.

Now a second offering: DVC/65, the best-yet version of C for Atari. It's by Ralph Walden, the author of ACE-C. This brand-new Freeware product is much more than just the C Development Compiler it claims to be. It's a whole programming system carefully crafted for the Atari, with its own special-purpose DOS, a superb editor, the most complete version of C for Atari which I have yet seen, and, best of all perhaps, 62 pages of very usable documentation. So far it has worked smoothly and easily, with convenient editing, fast assemblies, nearly automatic linking and respectable running times.

You get the normal C I/O functions, including scanf() — at last! — as well as printf(); special Atari floating point functions; an assembly language interface; great graphics utilities; RAMdisk for XE and Mosaic Board owners. Plus lots of sample C source programs (20+), demos, and special goodies for you MAC-65 and ACTION fans!

DVC/65 is issued as Freeware, which means the disk can be freely copied and distributed, while the documentation is coyrighted and available only from the author, for $35 (write to Ralph E. Walden, 1821 Jefferson, Eugene, OR 97402).

So there you have it: early notice about two promising additions to your Atari C library. Next time I'll tell you what I learn about both, warts and a'. C ya then!

— Dick Barkley

RESCUE MISSION

by Geoffrey Thompson

THE GAME:

You are flying an unarmed helicopter carrying vital medical supples through lengthy caverns in order to reach your agents who have been wounded behind enemy lines. The caverns are well protected with anti-aircraft batteries, space mines, rockets and lasers so the mission is dangerous and difficult and the caverns are lengthy. The only aid you have apart from your own skill is a radar scanner at the top of the screen to show your position in the cavern. The joystick controls the helicopter in any direction and you have three helicopters, each with a limited supply of fuel. The space bar will pause the game at any point.

TYPI"JG IT IN:

Type as listed. Make sure you save a copy of the program before you RUN it. Before you type Run, ensure you have a formatted disk in drive 1. The program will check the data line by line and write a file to disk with the filename 'D:RESCUE.DAT' and then ask you if you are ready to run. A 'Y' will begin the game. To RUN the program subsequently, type in the short program and save with whatever filename you wish on the same disk as 'D:RESCUE.DAT'. You can then RUN the program by typing RUN"D:(your filename)".

— Nora Young

DOS 2.5XL

(Reprint: Feedback, August, 1985, Adelaide, Australia)

When used with the XE computers, DOS 2.5 can make use of the extra memory available as a ramdisk. In addition, both DUP.SYS and MEM.SAV are stored on this ramdisk which allows instant access to DOS and automatic saving of any program in memory without a (normal) disk access.

This latter feature can be implemented on XL machines with 64k of memory by using the RAM behind the operating system ROMs. A program to do this with DOS 2.0S was published in number 24 of Analog magazine. The program presented here makes the same modification to DOS 2.5.

Type in the program and save it. Now RUN the program and, if you have typed it in correctly, you will get the message that the file PATCH25.OBJ has been created. If you did not boot your system using DOS 2.5, do so now. go to the DOS menu and select option L. In response to the file name request, type PATCH25.OBJ.

To save the patched version of DOS 2.5 use the H option. Now reboot your system using the disk containing your patched version of DOS and away you go.

— Alec Benson

DIGITAL LOGIC

(Reprint: BRACE, July, 1985)

Looking for something to do with your 520ST when . . . LOGO is the only language installed? Well, if you enjoy tinkering with electronics as I do, then you might enjoy this neat little project: A low cost, primitive (but effective) software circuit simulator. Due to time (and space) considerations this article will only outline the procedure and give one or two short examples. To expand this idea into a full-blown program is going to require a little ingenuity and work on your part, but the results should be worth it.

First of all, I should probably discuss just what a circuit simulator does. Loosely defined, it is a body of software which uses math (in this case boolean logic) to predict how an electronic circuit will behave. It is driven by inputs (let's see, if we apply a logic one here and a logic zero here . . .) and calculates the outputs of the circuit (. . . what should we read at this output pin?). With a well-designed simulator you can spend afew minutes with your Atari finding out if a new circuit design is going to work the way you think it will instead of hours wire-wrapping, perhaps in vain. Well, ok. There might be a slight exaggeration there. Anyway, now that you have an idea of what a simulator does, let's delve into how it works.

LOGO turns out to be a perfect language with which to implement a logic simulator because of the unique way in which it handles boolean logic. Where almost all other languages use zero to equal false and any number other than zero equals true, LOGO doesn't use numbers at all. TRUE and FALSE are special inputs used only with the boolean operators in LOGO. In addition, LOGO has what are known as predicates. These are special primitives, or procedures, which test for a specific condition. If the condition exists, a predicate will output TRUE, if not it will output FALSE. There are two other features of LOGO which make it attractive for this kind of application: It is procedural and recursive. We won't run into the advantages of recursion in the short examples I'll give here, but you'll soon see how the procedural nature of LOGO makes things "easy as pie."

The key to understanding how this simulator works is knowing what goes on inside an integrated circuit. Let's take an LS7408 chip for example. . . . The internal logic of this chip . . . is very simple. There are eight input pins arranged as four pairs. Each pair of input pins goes directly to he corresponding inputs of an AND gate. The output of each AND gate goes directly to one of the output pins of the chip. This may all seem extremely obvious to an experienced hobbyist, but it is necessary to look at each chip in this simplistic way. Each output pin must be defined in terms of the operation perfcrmed on the input pins. So, the output of pin three, for example, can be defined as being the result of pin one ANDed with pin two. In LOGO the definition of pin three can be something like this:

```
    TO LS7408.3
OUTPUT ( AND LS7408.1 LS7408.2 )
END
```

This is a procedure. When called, LS7408.3 will output the result of LS7408.1 ANDed with LS7408.2, which will also have been defined as procedures. In the above definition the OUTPUT command directs LOGO to return either a TRUE or FALSE to the calling procedure, depending upon the result of the AND operation. In accordance with LOGO syntax, the word AND comes first, followed by a list of its inputs. This is another advantage of LOGO: A boolean operation can have any number of inputs — not just two. The parentheses were optional in this instance. I just put them there for clarity.

Having defined output pin three in terms of its inputs, we can now try it out to see if it works. First, however, we'll need three more small procedures to tie up loose ends:

```
    TO LS7408.1
OUTPUT TRUE
END
    TO LS7408.2
OUTPUT FALSE
END
    TO TEST
PRINT LS7408.3
END
```

The first two procedures define the inputs to LS7408.3. The last one is necessary in order to both call it and to do something with the output. Simply type TEST at the LOGO prompt, and the word FALSE will be returned to you. You can try EDITing LS7408.1 and LS7408.2, changing the inputs to verify that everything workds properly. It will. In the above example, TRUE (or logic one) ANDed with FALSE (logic zero) should, and does return a FALSE. Neat, huh? Are the old cogs turning upstairs? Beginning to see how this can be useful? Well, let's go on, shall we?

Ok. So we have seen how to simulate a simple AND gate, but I still haven't shown why LOGO is so superior to other languages for something like this. You're probably saying to yourself, "Sure, it's nifty, but I could have written a two line BASIC program to do the same thing." True, but this is only one gate. Where LOGO really shines is in the combination of lots of gates. Let's take the example where the output pins three, six and eight of our LS7408 are all feeding inputs of a NAND gate in an LS7410 chip. Now what do we do?

Well, first off, we should define the other output pins of the LS7408:

```
    TO LS7408.6
OUTPUT ( AND LS7408.4 LS7408.5 )
END
    TO LS7408.8
OUTPUT ( AND LS7408.9 LS7408.10 )
END
```

Now to define the output of the NAND gate in the LS7410:

```
    TO LS7410.8
OUTPUT ( AND LS7410.3 LS7410.4 LS7410.5 )
END
```

EDIT TEST to reflect the new circuit.

```
    TO TEST
PRINT LS7410.6
END
```

Define the inputs to the 7408:

```
    TO LS7408.4
OUTPUT TRUE
END
    TO LS7408.5
OUTPUT TRUE
END
    TO LS7408.9
OUTPUT TRUE
END
    TO LS7408.10
OUTPUT TRUE
END
```

So, is that it? Whoops! We forgot to tell the software how the two chips are interconnected. Let's see, we could write a bunch of procedures defining the inputs of the LS7410 to be equal to the respective outputs of the LS7408 chip. Seems like a lot of work, huh? But wait! LOGO is a procedural language. Since the outputs of the LS7408 have been defined as procedures, and since procddures can be used as inputs to boolean operations in LOGO, all we have to do is a little EDITing of our definition of the output of the LS7410:

```
    TO LS7410.6
OUTPUT NOT ( AND LS7408.3 LS7408.6 LS7408.8 )
END
```

Now look at that. No, really. Sit back a second and look at what we just did. We defined the input pins of our triple input NAND gate to be the output pins of our three AND gates. Just to make sure the significance of this has sunk in, I'll outline what happens when you type TEST.

LOGO first tries to print LS7410.6 but finds it has unresolved inputs. The first of these inputs is LS7408.3. So LOGO has to see what the output of LS7408.3 is. Sure enough, the inputs to this procedure are also unresolved. So, once again, LOGO traces back another step to the inputs of LS7408.3 until it either finds another procedure with unresolved inputs or a TRUE or FALSE. It will then keep at this until all of the inputs to LS7410.6 are resolved and it can print out the result (which will be TRUE, of course — that one lousy little FALSE input at LS7408.2 screws us up. Change it to TRUE and see what happens). This all occurs in a split second, and all we see is the answer, which is all we really want. Now where's your little two line BASIC program? You can get BASIC to do this, of course, but not so easily or so elegantly.

Well, that's about all I'll say on the subject right now. Obviously there is more we could say. Entire circuits can be built using the guidelines laid out here. More complex chips, such as flip-flops, shift registers, etc., will require a little ingenuity to properly simulate. Just remember to define the output in terms of what happens to the inputs. Another area which needs attention is the inputs to the circuit. A procedure which takes patterns of ones and zeros and translates them into the TRUEs and FALSEs of the proper inputs is much preferred. But the thrust of this article is to get you started, and later, when you've mastered the concepts and programmed in all of those refinements, you can give me a call and I'll be right over with a blank disk . . .

— Jeff Griffen

Atari Computer Enthusiasts

A.C.E. is an independent, non-profit and tax exempt computer club and user's group with no connection to Atari Corp. We are a group interested in educating our members in the use of the Atari Computer and in giving the latest News, Reviews and Rumors.

All our articles, reviews and programs come from you, our members.

Our membership is world-wide; membership fees include the A.C.E. Newsletter. Dues are $14 a year for U.S., and $24 a year Overseas Airmail and include about 10 issues a year of the ACE Newsletter.

Subscription Dep't: 3662 Vine Maple Dr., Eugene, OR 97405.

**President— Dick Barkley, 2907 Wingate, Eugene, OR 97405 503-344-5843

Vice Pres— Larry Gold, 1927 McLean Blvd., Eugene, Or 97405 503-686-1490

8-bit Librarian— Chuck & Jody Ross, 2222 Ironwood, Eugene 97401 (503) 342-4133

ST Librarian— Jim Bumpas, 4405 Dillard Rd., Eugene OR 97405 503-484-4746

Editors— Mike Dunn, 3662 Vine Maple Dr., Eugene, OR 97405 503-344-6193

Jim Bumpas, 4405 Dillard Rd., Eugene, OR 97405 503-484-4746

E.R.A.C.E. (Education SIG Editor)— Nora Young, 105 Hansen Lane, Eugene, OR 97404 / 503-688-1458

Send 50c stamps or coin ($1 overseas) to the Ness' for the new, updated ACE Library List—new in May 85!

Bulletin Board
(503) 343-4352

On line 24 hours a day, except for servicing and updating. Consists of an 800 XL, 2 double-density double sided disk drives and 2 double-sided, double-density, 80-track disk drives, an Epson MX80 printer, a 1200 baud Prentice P212ST modem, running the Mindlink Bulletin Board software distributed by SofMark.

Best of ACE books

Volume 1 contains bound issues of the ACE Newsletter from the first issue, Oct 81 to June of 1982

Volume 2 covers July 1982 to June 1983

Only $12 each ($2 extra for Airmail). Available only from:
George Suetsugu
45-602 Apuapu St
Kanoehe, HI 96744

SortFinder Index

A composite index of Atari related articles in four popular computer periodicals, including ACE. Volume 1covers April, 1981 to June, 1983. Volume 2 covers July, 1983 to December, 1985. Only $6 per printed copy or $11 per disk copy for ACE members:

Jim Carr 2660 S.W. DeArmond
Corvallis, OR 97333

ATARI COMPUTER ENTHUSIASTS
3662 Vine Maple Dr. Eugene OR 97405

FIRST CLASS MAIL